Theoretical Computer Science 410 (2009) 1607–1615

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Strongly polynomial-time truthful mechanisms in one shot☆

Paolo Penna [b,*], Guido Proietti [c,d], Peter Widmayer [a]

[a] Institut für Theoretische Informatik, ETH, Zürich, Switzerland

[b] Dipartimento di Informatica ed Applicazioni "Renato M. Capocelli", Università di Salerno, Italy

[c] Dipartimento di Informatica, Università di L'Aquila, Italy

[d] Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti", CNR, Roma, Italy

ARTICLE INFO

Keywords:
Algorithmic game theory
Mechanism design
Design and analysis of algorithms
Minimum diameter spanning tree

ABSTRACT

One of the main challenges in *algorithmic mechanism design* is to turn (existing) efficient algorithmic solutions into efficient *truthful mechanisms*. Building a truthful mechanism is indeed a difficult process since the underlying algorithm must obey certain "monotonicity" properties and suitable *payment functions* need to be computed (this task usually represents the bottleneck in the overall time complexity).

We provide a general technique for building truthful mechanisms that provide optimal solutions in *strongly polynomial time*. We show that the *entire* mechanism can be obtained if one is able to express/write a strongly polynomial-time algorithm (for the corresponding optimization problem) as a "suitable combination" of simpler algorithms. This approach applies to a wide class of *mechanism design graph problems*, where each selfish agent corresponds to a weighted edge in a graph (the weight of the edge is the cost of using that edge). Our technique can be applied to several optimization problems which prior results cannot handle (e.g., MIN–MAX optimization problems).

As an application, we design the first (strongly polynomial-time) truthful mechanism for the *minimum diameter spanning tree* problem, by obtaining it directly from an existing algorithm for solving this problem. For this non-utilitarian MIN–MAX problem, no truthful mechanism was known, even considering those running in exponential time (indeed, exact algorithms do not necessarily yield truthful mechanisms). Also, standard techniques for payment computations may result in a running time which is not polynomial in the size of the input graph. The overall running time of our mechanism, instead, is polynomial in the number n of nodes and m of edges, and it is only a factor $O(n\alpha(n, n))$ away from the best known canonical centralized algorithm.

1. Introduction

The emergence of the Internet as the platform for distributed computing has posed interesting questions on how to design efficient solutions which account for the lack of a "central authority" [11,15,16]. This aspect is certainly a key ingredient for the success of the Internet and, probably, of any "popular" system that one can envision (peer-to-peer systems are a notable example of this type of anarchic systems). In their seminal works, Koutsoupias and Papadimitriou [11] and Nisan and Ronen

☆ Work partially supported by the Research Project GRID.IT, funded by the Italian Ministry of Education, University and Research, by the European Project IST-15964 "Algorithmic Principles for Building Efficient Overlay Computers" (AEOLUS), by the European Union under COST 295 (DYNAMO), and by the Swiss BBW. Part of this work has been developed while the first and the second author were visiting ETH. A preliminary version of this paper was presented at the *2nd International Workshop on Internet and Network Economics (WINE)*, December 15–17, 2006, Patra, Greece.

* Corresponding author.
E-mail addresses: penna@dia.unisa.it (P. Penna), proietti@di.univaq.it (G. Proietti), widmayer@inf.ethz.ch (P. Widmayer).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.12.034

[15], suggest a *game-theoretic* approach in which the various "components" of the system are modeled as *selfish agents*: each agent performs a "strategy" which results in the highest *utility* for him/her-self. For instance, each agent may control a link of a communication network and each link has a cost for transmitting (i.e., for using it). A protocol that wishes to establish a minimum-cost path between two nodes would have to ask the agents for the cost of the corresponding link [15, 2]. An agent may thus find it to be in his/her interest to lie about his/her costs (e.g., an agent might untruthfully report a very high cost in order to induce the protocol to use an alternative link, and thus no cost for the agent). Nisan and Ronen [15] propose a *mechanism design* approach that combines an underlying algorithm (e.g., a shortest path algorithm) with a suitable payment function (e.g., how much we pay an agent for using his/her link). The idea is to come up with a so called *truthful mechanism*, that is, a combination of an algorithm with payments which guarantee that no agent can improve his/her own utility by misreporting his/her piece of private information (e.g., the cost of his/her link). Unfortunately, the design of truthful mechanisms is far from trivial and known results, originally developed in the microeconomics field [20,3,5,13], pose new algorithmic challenges which are the main subject of *algorithmic mechanism design* (see e.g. [4]).

Some interesting classes of problems (including a family of mechanism design graph problems considered here and in a number of works [15,7,6,10]) require the underlying algorithm to be *monotone* (e.g., if the algorithm selects an edge then it cannot drop this edge if its cost gets smaller and everything else remains the same). Though this condition suffices for the existence of a truthful mechanism [13,2], it is not clear how to guarantee this property nor how the corresponding payment functions can be efficiently computed (see e.g. [9,14]).

Mu'Alem and Nisan [12] were the first to propose a general method for constructing monotone algorithms (and thus truthful mechanisms). Basically, their approach consists of a set of "rules" to combine monotone algorithms so that the final combination results in a monotone algorithm as well. As observed by Kao et al. [10], the method in [12] does not provide an *efficient* way of computing the payments. Kao et al. [10] then extend some of the techniques in [12] and provide an efficient way for computing the corresponding payment functions. Kao et al.'s approach [10] represents a significant progress towards a general technique which accounts for computational issues, though it cannot be applied to some very basic graph problems (e.g., a problem recently tackled in [18] — we discuss this issue more in detail below).

1.1. Our contribution

In this work, we turn one of the main results in [12] into a general technique for building optimal truthful mechanisms running in *strongly polynomial time* (optimality refers to the quality of the computed solution). We show that the *entire* mechanism can be obtained if one is able to express/write an algorithm (for the corresponding optimization problem) as a "suitable combination" of simpler ones (see Section 2 and Theorem 1 therein). Obviously, the resulting mechanism is optimal and/or runs in strongly polynomial time if the algorithm does. However, neither of these conditions is required by our technique to guarantee truthfulness. This approach applies to a wide class of *mechanism design graph problems*, where each selfish agent corresponds to a weighted edge in a graph (the weight of the edge is the cost of using that edge). Our technique can deal with problems in which the cost function "underlying" the algorithm(s) is any *monotonically non-decreasing* function in the edge weights of the graph (i.e., in the costs of the agents). Since this includes several *non-utilitarian*[1] problems (e.g., MIN–MAX optimization functions), the results in [12] extend "only partially", that is, truthfulness can be guaranteed but the payments computation cannot be done "directly" by computing the "alternative" solution in which an agent is removed from the input (see e.g. [15,9]). We indeed observe that, for the problems considered in this work (see the discussion in Example 1), the payments computation is more complex than the case of *monotonically increasing* optimization functions, which are assumed in both [12] (where the problem is utilitarian) and in [10] (this assumption precedes Theorem 10 in [10] and the applications therein consist exclusively of utilitarian graph problems).

In Section 3, we apply our technique to the *minimum diameter spanning tree* problem and obtain the first (strongly polynomial-time) mechanism for it. For this non-utilitarian MIN–MAX problem, no truthful mechanism was known, even considering those running in exponential time (indeed, exact algorithms do not necessarily yield truthful mechanisms — see Fact 1). Also, standard techniques for payment computations may result in a running time which is not polynomial in the size of the input graph (see discussion in Example 1). The overall running time of our mechanism is instead $O(mn^2\,\alpha(n, n))$, and thus is only a factor $O(n\,\alpha(n, n))$ away from the best known algorithm for this problem [8], where $\alpha(\cdot, \cdot)$ is the classic inverse of the Ackermann's function. For two-edge connected graphs we also guarantee the *voluntary participation* condition, that is, no truthful agent runs into a loss (see next section for a formal definition). The minimum diameter spanning tree has both theoretical and practical relevance (e.g., in a peer-to-peer system we may want to set up a loop-free logical network using the resources – links – of a physical network so that any two peers can communicate efficiently).

The results for the minimum diameter spanning tree are paradigmatic of what happens when considering certain non-utilitarian mechanism design problems (another case is the *minimum radius spanning tree* problem [18] described in Example 1). First, one has to determine whether an existing algorithm can be turned into a truthful mechanism, whether a new one is needed, or if none can serve for this purpose [15,2,18]. In case a suitable algorithm exists, one has to find out how to compute the corresponding payments efficiently, possibly without burdening the complexity of the chosen

[1] An optimization problem is called utilitarian if the goal is to minimize the sum of all agents costs or, equivalently, to maximize the sum of all agents valuations. Utilitarian graph problems have been studied in [15,9,10].

algorithm [9,18]. Our technique can be used to give a positive answer to both questions, and thus to obtain the efficient mechanism in "one shot" (see Theorem 1). Besides that, we discuss other possible extensions and applications of our technique in Section 4, where the concluding remarks are given.

1.2. Mechanism design graph problems

Consider problems in which we are given a graph $G = (V, E)$ and the set of feasible outcomes consists of a suitable set $\mathcal{O} = \mathcal{O}(G)$ which depends only on the combinatorial structure of the graph (e.g., it consists of certain subgraphs of G). We have one agent per edge and the type $t_e \in \mathbb{R}^+$ of agent e is nothing but the *weight* of edge $e \in E$. Each solution $Y \in \mathcal{O}$ *uses* a subset of the edges of G; in particular, if Y uses edge e, then agent e has a cost (for implementing this outcome) equal to t_e. This scenario is common to several problems considered in the algorithmic mechanism design community: shortest path [15], minimum spanning tree [15], shortest path tree [7], minimum-radius spanning tree [18]. Consider an agent e and let $\mathbf{r}_{-e}$ denote the values reported by the other agents, that is, $\mathbf{r}_{-e} = (r_1, \ldots, r_{e-1}, r_{e+1}, \ldots, r_m)$. When agent e reports x and the other agents report $\mathbf{r}_{-e}$, algorithm A computes a feasible outcome $A(x, \mathbf{r}_{-e})$. (That is, the algorithm returns a solution on input the vector $(x, \mathbf{r}_{-e}) := (r_1, \ldots, r_{e-1}, x, r_{e+1}, \ldots, r_m)$.) We say that declaration x is a *winning declaration* if solution $A(x, \mathbf{r}_{-e})$ uses edge e. A *mechanism* $M = (A, P)$ associates a payment $P_e(x, \mathbf{r}_{-e})$ with every agent e whose declaration x is a winning declaration (given the other agents' declarations $\mathbf{r}_{-e}$). This determines the *utility* of agent e:

$$u_e^M(x, \mathbf{r}_{-e}) := \begin{cases} P_e(x, \mathbf{r}_{-e}) - t_e & \text{if } A(x, \mathbf{r}_{-e}) \text{ uses } e, \\ 0 & \text{otherwise.} \end{cases}$$

Mechanism M is a *truthful* mechanism (with dominant strategies) if every function $u_e^M(x, \mathbf{r}_{-e})$ is maximized for $x = t_e$, for all $\mathbf{r}_{-e}$. We are interested in truthful mechanisms which optimize some objective function $\mu(Y, \mathbf{t})$ depending on the agents types $\mathbf{t} = (t_1, \ldots, t_m)$. Notice that the mechanism will work on the reported types $\mathbf{r}$. Hence, truthfulness guarantees that, if the algorithm returns an optimal solution for the given input, then the mechanism outputs an optimal solution w.r.t. the true types. We will also consider mechanisms which satisfy the *voluntary participation*, that is, a truthful agent is guaranteed to have a non-negative utility (i.e., $u_e^M(t_e, \mathbf{r}_{-e}) \geq 0$). This property will be achieved whenever there exists an "alternative" solution that does not use edge e, i.e., $\mathcal{O}(G - e) \neq \emptyset$.

2. A technique for efficient truthful mechanisms

Our approach consists in defining an optimal algorithm A as a "suitable combination" of simpler ones. For minimization problems, we combine algorithms by means of the following 'MIN' operator, which is essentially the same as the 'MAX' operator by Mu'Alem and Nisan [12]:

$\mathbf{MIN}_\mu(A_1, A_2)$ operator
- compute $Y_1 = A_1(\mathbf{r})$ and $Y_2 = A_2(\mathbf{r})$;
- if $\mu(Y_1, \mathbf{r}) \leq \mu(Y_2, \mathbf{r})$ then return Y_1 else return Y_2.

We can recursively apply this operator to several algorithms and obtain a new one:

$$\mathrm{MIN}_\mu(A_1, \ldots, A_k) := \mathrm{MIN}_\mu(\mathrm{MIN}_\mu(A_1, \ldots, A_{k-1}), A_k).$$

Notice that the ordering among the algorithms specifies how the new algorithm breaks ties. Our main concern is to have a general technique for building truthful mechanisms which optimize $\mu(\cdot)$ and that are *computationally efficient*.

To this end, we will assume that each algorithm A_i satisfies a property (called *plateau-like*) which is slightly stronger than the one (called bitonic) used in [12]:

Definition 1 (*Plateau-like Algorithm*). An algorithm A for a mechanism design graph problem is *monotone* if, for all agents e, and for all $\mathbf{r}_{-e}$ there exists a threshold $\theta_e(\mathbf{r}_{-e}) \in (\mathbb{R}^+ \cup \infty)$ such that (i) every $x \leq \theta_e(\mathbf{r}_{-e})$ is a winning declaration and (ii) every $x > \theta_e(\mathbf{r}_{-e})$ is not a winning declaration. A monotone algorithm A is *plateau-like* w.r.t. $\mu(\cdot)$ if, for all e, for all $\mathbf{r}_{-e}$, the function $g_A(x) := \mu(A(x, \mathbf{r}_{-e}), (x, \mathbf{r}_{-e}))$ is non-decreasing in x and constant for $x > \theta_e(\mathbf{r}_{-e})$.

It is well known that an algorithm A can be turned into a truthful mechanism (A, P) if and only if A is monotone [13,2], in which case the payments are uniquely[2] determined by the thresholds:

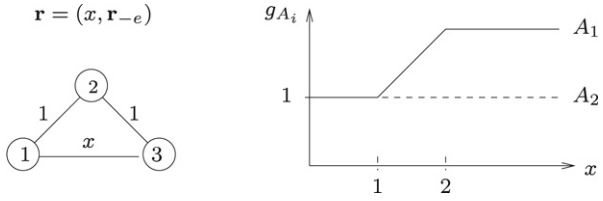
$$P_e(x, \mathbf{r}_{-e}) = \begin{cases} \theta_e(\mathbf{r}_{-e}) & \text{if } A(x, \mathbf{r}_{-e}) \text{ uses } e, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Mu'Alem and Nisan [12] proved that, if all algorithms A_i are bitonic, then the algorithm $A = \mathrm{MIN}_\mu(A_1, \ldots, A_k)$ is monotone, and thus truthfulness can be guaranteed. Our main contribution here is a method for computing these payments efficiently

[2] If $\theta_e(\mathbf{r}_{-e}) = \infty$, then we can set the payment of e to be any constant value and guarantee truthfulness. This case arises if edge e will be always included by A, e.g. for $\mathcal{O}(G - e) = \emptyset$, in which case voluntary participation cannot be guaranteed (unless we assume an upper bound on t_e). Otherwise, i.e. $\theta_e(\mathbf{r}_{-e}) < \infty$, the only payments which guarantee truthfulness are those in (1) [12] which then satisfy the voluntary participation condition.

if we assume that the algorithms are plateau-like. This task is non-trivial since the computation of the thresholds of a 'MIN' combination of algorithms can be rather involved if $\mu(\cdot)$ is not monotone increasing in x as in [12,10]:

Example 1 (*Minimum Radius Spanning Tree (MRST)*). Consider the problem of computing the *minimum radius spanning tree*, that is, a spanning tree rooted at some node of the graph whose height is minimal. Consider the following simple graph (left):



If A_i outputs a shortest path tree rooted at node i and $h(\cdot)$ denotes the height of any rooted tree, then both $A := \mathrm{MIN}_h(A_1, A_2)$ and $A' := \mathrm{MIN}_h(A_2, A_1)$ compute a MRST for this graph. However, the thresholds $\theta_e(\mathbf{r}_{-e})$ and $\theta'_e(\mathbf{r}_{-e})$ of the two algorithms are different. This is due to a different tie-breaking rule: For $0 \le x \le 1$, algorithms A_1 and A_2 have the same cost, i.e., $g_{A_1}(x) = g_{A_2}(x)$; hence,

$$A(x, \mathbf{r}_{-e}) = \begin{cases} A_1(x, \mathbf{r}_{-e}) & \text{if } x \le 1 \\ A_2(x, \mathbf{r}_{-e}) & \text{otherwise} \end{cases}$$

while $A'(x, \mathbf{r}_{-e}) = A_2(x, \mathbf{r}_{-e})$. Since A_2 never uses edge e, while A_1 uses this edge for $x \le 2$, it turns out that $\theta_e(\mathbf{r}_{-e}) = 1$ and $\theta'_e(\mathbf{r}_{-e}) = 0$.

Observe that, the threshold of algorithm $\mathrm{MIN}_h(A_1, A_2)$ is *different from the thresholds of the two algorithms*. Its computation depends on the way the functions g_{A_i} cross with each other, which in general can be quite involved (we have to consider how n "stairway" functions intersect pairwise [18] and the order in which we break ties). Finally, a binary search of this threshold may require a time which *depends on the edge weights* (namely, the logarithm of the largest reported type) and thus *not* strongly polynomial time, i.e., not polynomial in the number of nodes and edges. □

We reduce the computation of the payment $P_e(x, \mathbf{r}_{-e})$ to the task of computing, for every algorithm A_i, three thresholds $\theta^i = \theta^i_e(\mathbf{r}_{-e})$, $\hat{\theta}^i = \hat{\theta}^i_e(\mathbf{r}_{-e})$ and $\check{\theta}^i = \check{\theta}^i_e(\mathbf{r}_{-e})$. The value θ^i is the threshold in Definition 1 relative to algorithm A_i. The other two thresholds are defined as follows. Since A_i is plateau-like, $g_{A_i}(x) = \mu(A_i(x, \mathbf{r}_{-e}), (x, \mathbf{r}_{-e}))$ is constant for all $x > \theta^i$, where it also reaches its maximum. Let $\overline{g}_i$ be this maximum and let $g_{\min} := \min_i\{\overline{g}_i\}$. We let $\inf\{\emptyset\} = \infty$ and define $\hat{\theta}^i, \check{\theta}^i \in (\mathbb{R}^+ \cup \infty)$ as follows:

$$\hat{\theta}^i := \inf\{x |\, g_{A_i}(x) \ge g_{\min}\}; \tag{2}$$

$$\check{\theta}^i := \inf\{x |\, g_{A_i}(x) > g_{\min}\}. \tag{3}$$

Notice that the maximum $\overline{g}_i$ can be easily computed knowing θ_i. This is the main "additional" feature of plateau-like algorithms over bitonic ones. Intuitively speaking, $g_{\min}$ is the minimum cost if we do *not* use edge e. Thus, the solution of algorithm A_i will be selected only if its cost is better/not worse than this value (depending on the used tie-breaking rule). The two thresholds in (2) and (3) say what is the largest x for which this happens.

Our general approach for constructing computationally efficient mechanisms consists in rewriting algorithms as suggested by the following:

Definition 2 (*MIN-reducible Algorithm*). An algorithm A is *MIN-reducible* if it can be written as the 'MIN' of plateau-like algorithms. That is, there exist k algorithms $A_1, \ldots, A_k$ such that $A = \mathrm{MIN}_\mu(A_1, \ldots, A_k)$ and each algorithm A_i is plateau-like w.r.t. $\mu(\cdot)$. Such an algorithm A is *MIN-reducible in τ time* if, for every input $\mathbf{r}$, it is possible to compute all thresholds $\theta^i_e(\mathbf{r}_{-e})$, $\hat{\theta}^i_e(\mathbf{r}_{-e})$ and $\check{\theta}^i_e(\mathbf{r}_{-e})$ in at most τ time steps, for all $1 \le i \le k$ and for all edges e used by $A(\mathbf{r})$.

The following result provides a powerful tool for designing efficient truthful mechanisms:

Theorem 1. *If algorithm A is MIN-reducible in $O(\tau)$ time, then there exist payment functions P such that (A, P) is a truthful mechanism and all payments $P_e(x, \mathbf{r}_{-e})$ can be computed in $O(\tau + k(\tau_\mu + N))$ time, where τ_μ is the time to compute $\mu(\cdot)$ and N is the number of used agents/edges.*

Proof. The first part of the theorem follows from a result by Mu'Alem and Nisan [12]. In order to prove the second part, we simply show that, given the values $\theta^i, \hat{\theta}^i$ and $\check{\theta}^i$, it is possible to compute $\theta_e(\mathbf{r}_{-e})$ in $O(k)$ time after the following preprocessing requiring $O(k \cdot \tau_\mu)$ time. First of all, we compute the index $i_{\min}$ of the first algorithm A_i such that $\overline{g}_i = g_{\min}$. This requires $O(k \cdot \tau_\mu)$ time for computing all $\overline{g}_i$, and from that the computation of $g_{\min}$ and $i_{\min}$ requires $O(k)$ time. (Recall that $\overline{g}_i = g_{A_i}(x)$ for any $x > \theta^i$.) Then, we prove the following:

$$\theta_e(\mathbf{r}_{-e}) = \max\{\theta^{i_{\min}}, \max\{\hat{\theta}^i |\, i > i_{\min}\}, \max\{\check{\theta}^i |\, i < i_{\min}\}\}, \tag{4}$$

which we do by first proving that:

$$\theta_e(\mathbf{r}_{-e}) \geq \theta^{i_{\min}} \tag{5}$$

$$\theta_e(\mathbf{r}_{-e}) \geq \breve{\theta}^i \quad \text{if } i < i_{\min} \text{ and } \breve{\theta}^i > 0 \tag{6}$$

$$\theta_e(\mathbf{r}_{-e}) \geq \hat{\theta}^i \quad \text{if } i > i_{\min} \text{ and } \hat{\theta}^i > 0. \tag{7}$$

If (5) would not hold, then there would exist $x < \theta^{i_{\min}}$ such that $A(x, \mathbf{r}_{-e})$ does not use edge e and, by definition of 'MIN', $A(x, \mathbf{r}_{-e}) = A_i(x, \mathbf{r}_{-e})$ for some $i \neq i_{\min}$. We can also assume w.l.o.g. that $x > \theta^i$, thus implying $g_A(x) = \mu(A(x, \mathbf{r}_{-e}), (x, \mathbf{r}_{-e})) = g_{A_i}(x) = \mu(A_i(x, \mathbf{r}_{-e}), (x, \mathbf{r}_{-e})) = \overline{g}_i$. Since $\theta^i < \theta^{i_{\min}}$, and by definition of $g_{\min}$ and $i_{\min}$, $\overline{g}_i > g_{\min} = \mu(A_{i_{\min}}(x, \mathbf{r}_{-e}), (x, \mathbf{r}_{-e}))$. This contradicts the definition of 'MIN': for the input $(x, \mathbf{r}_{-e})$, algorithm $A_{i_{\min}}$ returns a solution of cost strictly better than that by algorithm A_i, and thus A cannot prefer $A_i(x, \mathbf{r}_{-e})$ to $A_{i_{\min}}(x, \mathbf{r}_{-e})$.

If (6) would not hold, then we could pick an x other than any θ^l, and such that $\theta_e(\mathbf{r}_{-e}) < x < \breve{\theta}^i$. Algorithm A_i returns a solution of cost

$$\mu(A_i(x, \mathbf{r}_{-e}), (x, \mathbf{r}_{-e})) = g_{A_i}(x) \leq g_{\min}.$$

Since $x > \theta_e(\mathbf{r}_{-e})$, algorithm A returns a solution $A_j(x, \mathbf{r}_{-e})$ which does not use edge e. This, and the fact that $x \neq \theta^j$, imply that $x > \theta^j$. Hence the cost of this solution is $\mu(A_j(x, \mathbf{r}_{-e}), (x, \mathbf{r}_{-e})) = g_{A_j}(x) = \overline{g}_j \geq g_{\min}$. Inequality (5) and the definition of 'MIN' imply that $\overline{g}_j = g_{\min}$ and therefore $j = i_{\min}$. This contradicts the definition of 'MIN': since $i < i_{\min} = j$, for the input $(x, \mathbf{r}_{-e})$, algorithm A should prefer solution $A_i(x, \mathbf{r}_{-e})$ to solution $A_j(x, \mathbf{r}_{-e})$.

If (7) would not hold, then we could pick an x as in the previous case such that $\theta_e(\mathbf{r}_{-e}) < x < \hat{\theta}^i$. Algorithm A_i returns a solution of cost

$$\mu(A_i(x, \mathbf{r}_{-e}), (x, \mathbf{r}_{-e})) = g_{A_i}(x) < g_{\min}.$$

Similarly to the previous case, algorithm A returns instead a solution $A_j(x, \mathbf{r}_{-e})$ which does not use edge e. Hence, $\mu(A_j(x, \mathbf{r}_{-e}), (x, \mathbf{r}_{-e})) = g_{A_j}(x) = \overline{g}_j \geq g_{\min}$. This again contradicts the definition of 'MIN' since algorithm A would prefer solution $A_j(x, \mathbf{r}_{-e})$ to solution $A_i(x, \mathbf{r}_{-e})$ which has a strictly better cost.

Now, in order to prove (4), we show that one out of (5)–(7) must hold with '='. By way of contradiction, assume this is not the case. Then we could pick an x other than any θ^i such that

$$\theta_e(\mathbf{r}_{-e}) > x > \theta^{i_{\min}} \tag{8}$$

$$\theta_e(\mathbf{r}_{-e}) > x > \breve{\theta}^i \quad \text{if } i < i_{\min} \text{ and } \breve{\theta}^i > 0 \tag{9}$$

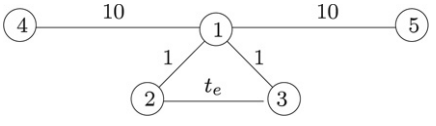
$$\theta_e(\mathbf{r}_{-e}) > x > \hat{\theta}^i \quad \text{if } i > i_{\min} \text{ and } \hat{\theta}^i > 0. \tag{10}$$

On input $(x, \mathbf{r}_{-e})$ algorithm A returns a solution $A_i(x, \mathbf{r}_{-e})$, for some i. Inequality (8) implies that $A_i(x, \mathbf{r}_{-e})$ uses edge e and therefore $i \neq i_{\min}$. If $i < i_{\min}$ then (9) implies $\mu(A_i(x, \mathbf{r}_{-e}), (x, \mathbf{r}_{-e})) = g_{A_i}(x) = \overline{g}_i > g_{\min}$. This contradicts the definition of 'MIN' since $A_{i_{\min}}(x, \mathbf{r}_{-e})$ would be better. If $i > i_{\min}$, then (9) implies $\mu(A_i(x, \mathbf{r}_{-e}), (x, \mathbf{r}_{-e})) = g_{A_i}(x) = \overline{g}_i \geq g_{\min}$. In this case, A should have preferred algorithm $i_{\min}$ to algorithm i, because of $i > i_{\min}$.

Obviously, if we know $\hat{\theta}^i$, $\breve{\theta}^i$ and $i_{\min}$, then equality (4) says that a single $\theta_e(\mathbf{r}_{-e})$ can be computed in time linear in k. From (1) we need to compute the payments only for the N edges used in $A(\mathbf{r})$. In this way, by Definition 2, the overall computation of all such $P_e(x, \mathbf{r}_{-e})$ takes $O(\tau + k \cdot \tau_\mu + k \cdot N)$ time. □

3. The minimum diameter spanning tree problem

In the *minimum diameter spanning tree* (MDST) problem we are given a weighted undirected graph and the goal is to find a spanning tree which minimizes the longest path between any two nodes in that tree (the length of a path is the sum of the weights of its edges). In this section we study the corresponding mechanism design graph problem. Formally, given a graph $G = (V, E)$, the set $\mathcal{O}(G)$ of feasible solutions consists of all spanning trees; the set of used edges naturally consists of all edges in the tree, and the goal is to find a tree T of minimum *diameter*, that is, a tree such that the length of a maximum-length simple path in T is minimum. We denote this value by $d(G, \mathbf{t})$. Consider the following graph:



For all $t_e \leq 9$, *any* spanning tree is a MDST since, according to the edge weights $\mathbf{t}$ in the picture, the maximum-length simple path is the upper one and this path appears in any spanning tree. Unfortunately, the fact that an algorithm is exact for the MDST problem is not sufficient for obtaining a truthful mechanism. A well-known result by Myerson [13] (see also Archer and Tardos [2]) states that, for our problem, truthfulness can be achieved only if the algorithm is monotone (see Definition 1). Actually, it is possible to show that exact algorithms need not lead to truthful mechanisms:

Fact 1. *Let A be an exact algorithm which, for the instance above, returns a tree containing edge e if and only if* $r_e \in [0, 5] \cup [7, 9]$. *(We let* $\mathbf{r}_{-e} = \mathbf{t}_{-e}$ *being the numbers shown in the picture.) Then there exist no payment functions P such that* (A, P) *is truthful for the problem instance above.*

In what follows, we will show that there exists an efficient polynomial-time algorithm for the MDST problem which is monotone and such that the payments can be computed efficiently. Both results follow from our main technique (Theorem 1).

3.1. A MIN-reducible algorithm for the MDST problem

The computation of a MDST of a given graph can be reduced to the computation of a shortest path tree rooted at the *absolute* 1-*center* (simply *center*, in the following) of G [8]. Loosely speaking, the center of a graph is a point c located on an edge (or on one of its endpoints) such that the distance from c to the farthest node is minimized. In particular, all edges are rectifiable, meaning that any point c on edge $f = (u, v)$ can be specified as a pair $c = (f, \lambda)$ with $\lambda \in [0, 1]$; in this case, we obtain a new graph G_c where edge (u, v) is replaced by two edges (u, c) and (v, c); their weights are $\overline{uc} := \lambda t_e$ and $\overline{vc} := (1 - \lambda)t_e$, respectively. (Notice that we consider each edge as an ordered pair of vertices.) Given a point c on f, one can build a spanning tree T_c of G by computing a shortest path tree of G_c rooted at c, and then by replacing edges incident to c with the edge (u, v). Trivially, the tree T_c has diameter at most $2h_f^\lambda(\mathbf{t})$.[3] We let $h_f^*(\mathbf{t})$ be the minimum height among all shortest paths trees rooted at some point on f, that is, $h_f^*(\mathbf{t}) := \min_{\lambda \in [0,1]} h_f^\lambda(\mathbf{t})$. Our building block is the following algorithm which computes the *relative center* of edge f for the reported input $\mathbf{r}$, namely, a point $c = (f, \lambda)$ minimizing $h_f^\lambda(\mathbf{r})$:

Algorithm CENTER$_f$

1. compute the minimum $\lambda \in [0, 1]$ such that $h_f^\lambda(\mathbf{r}) = h_f^*(\mathbf{r})$;
2. compute the tree T_c for $c = (f, \lambda)$ and edge weights $\mathbf{r}$;
3. return $Y = (T_c, c)$. /* return the tree T_c associated with the SPT and the center */

Since it holds that $d(G, \mathbf{r})/2 = h^*(\mathbf{r}) := \min_f h_f^*(\mathbf{r})$ [8], we can compute a MDST by searching through all relative centers of G for a best possible position of the center:

$$A_{\text{MDST}} := \text{MIN}_h(\text{CENTER}_{e_1}, \ldots, \text{CENTER}_{e_m}),$$

where $e_1, \ldots, e_m$ denote the edges of G in some arbitrary order (independent of the agents' bids). We stress that a MDST cannot be obtained by restricting the computation of the relative center to one of the endpoints of edge f, that is, by considering only the vertices as possible center locations. This will produce a minimum radius spanning tree, instead, and thus the mechanism in [18] cannot be used here.

The following result, combined with Theorem 1, implies the existence of a truthful mechanism for the MDST:

Theorem 2. *Algorithm* A_{MDST} *is MIN-reducible and, on input a graph G with edge weights* $\mathbf{r}$*, it returns a MDST and an absolute center for this input. This computation requires* $O(mn\,\alpha(n, n))$ *time.*

Proof. In order to build a truthful mechanism out of Algorithm A_{MDST}, we will consider a slightly more general formulation which includes both the MDST and the absolute center problems. In particular, we consider the set $\mathcal{O}(G)$ as all pairs $Y = (T, c)$, where T is a spanning tree of G and $c = (f, \lambda)$ is the position of the center, which must be located on some of the edges in T. For any solution $Y = (T, c) \in \mathcal{O}(G)$, we let $h(Y, \mathbf{t})$ be the maximum distance from c to any node in the tree T, i.e., the height of the tree T when considering c as the root. We say that solution $Y = (T, c)$ uses edge e if $e \in T$.

Algorithm A_{MDST} *is MIN-reducible.* We need to prove that every algorithm CENTER$_f$ is plateau-like w.r.t. $h(\cdot)$. We first prove that algorithm CENTER$_f$ is monotone. Consider $\mathbf{r} = (r_e, \mathbf{r}_{-e})$ and $\mathbf{r}' = (r'_e, \mathbf{r}_{-e})$, with $r'_e > r_e$. Let $Y = (T, (f, \lambda)) = \text{CENTER}_f(\mathbf{r})$ and $Y' = (T', (f, \lambda')) = \text{CENTER}_f(\mathbf{r}')$. By definition, $h_f^\lambda(\mathbf{r}) = h_f^*(\mathbf{r}) \le h_f^*(\mathbf{r}') = h_f^{\lambda'}(\mathbf{r}')$, where the inequality follows from the observation that $h_f^*(\cdot)$ is monotonically non-decreasing in the edge weights. By contradiction, assume that Y' uses edge e, while Y does not. Then the tree T does not contain edge e and its height remains $h_f^\lambda(\mathbf{r})$ also when the edge weights are $\mathbf{r}'$. Hence, $h_f^*(\mathbf{r}') \le h_f^\lambda(\mathbf{r}) = h_f^*(\mathbf{r})$, thus implying $h_f^*(\mathbf{r}) = h_f^*(\mathbf{r}')$. The first step of CENTER$_f$ yields $\lambda = \lambda'$. Hence, if we restrict ourselves to inputs $\mathbf{r}$ and $\mathbf{r}'$, algorithm CENTER$_f$ returns the set of edges of a shortest path tree on G_c with source c and edge weights $\mathbf{r}$ or $\mathbf{r}'$. Since shortest path tree algorithms are monotone [7,10], we contradict the hypothesis that CENTER$_f$ violates the monotonicity for inputs $\mathbf{r}$ and $\mathbf{r}'$. To show that CENTER$_f$ is plateau-like we observe that the function $h(\text{CENTER}_f(x, \mathbf{r}_{-e}), (x, \mathbf{r}_{-e})) = h_f^*(x, \mathbf{r}_{-e})$ is non-decreasing in x and constant for all values x for which $\text{CENTER}_f(x, \mathbf{r}_{-e})$ does not use edge e.

Correctness. On input of a graph G with edge weights $\mathbf{r}$, the algorithm returns a MDST and an absolute center for this input. The proof of this fact is due to [8].

[3] Formally, the tree T_c is obtained by removing c from the shortest path tree and by adding back edge (u, v), unless c is sitting on one of the endpoints of (u, v) and is not connected to the other endpoint.

Complexity. We compute the distances $\delta_{u,v}(G, \mathbf{r})$, for all nodes u and v. Using the algorithm by Pettie and Ramachandran [17], this step takes $O(mn \log \alpha(m, n))$ time. The running time then follows from this result:

Lemma 1. *Given all distances $\delta_{u,v}(G, \mathbf{r})$, for all nodes u and v, the computations of algorithm CENTER$_f$ can be performed in $O(n\,\alpha(n, n))$ time.*

Proof. Recall that any point on edge f is represented as pair (f, λ), with $\lambda \in [0, 1]$. The distance from the point (f, λ) to another node z in $(G, \mathbf{r})$ is

$$\delta_{(f,\lambda),z}(G, \mathbf{r}) = \min\{\lambda r_f + \delta_{u,z}(G, \mathbf{r}), (1 - \lambda)r_f + \delta_{v,z}(G, \mathbf{r})\}. \tag{11}$$

Observe that $h_f^\lambda(\mathbf{r}) = \max_{z \in V} \delta_{(f,\lambda),z}(G, \mathbf{r})$. Define functions $f_z(\lambda) := \delta_{(f,\lambda),z}(G, \mathbf{r})$, for $\lambda \in [0, 1]$. According to the first step of CENTER$_f$, we have to compute the smallest $\lambda^* \in [0, 1]$ which is a minimum for the *upper envelope* of all functions in $\{f_z(\lambda)\}_{z \in V}$, i.e., $\max_{z \in V} f_z(\lambda^*) = \min_{\lambda \in [0,1]} \max_{z \in V} h_f^\lambda(\mathbf{r}) = h_f^*(\mathbf{r})$. Since the functions in $\{f_z(\lambda)\}_{z \in V}$ intersect pairwise in at most one point (see Eq. (11)), this computation requires $O(n\,\alpha(n, n))$ time [1]. □

Since there are m algorithms, the overall complexity of A_{MDST} is $O(mn\,\alpha(n, n))$, which dominates the $O(mn \log \alpha(m, n))$ factor for the initial computations. □

We need one more step to guarantee that payments can be computed in strongly polynomial time. One of our major technical contributions is to show that the "MIN-reduction" can be done efficiently:

Theorem 3. *Algorithm A_{MDST} is MIN-reducible in $O(mn^2\,\alpha(n, n))$ time.*

Proof. We compute the distances $\delta_{u,v}(G, \mathbf{r})$ and $\delta_{u,v}(G{-}e, \mathbf{r}_{-e})$, for all nodes u and v, and for all edges e used by the computed solution. Using the $O(mn \log \alpha(m, n))$-time all-pairs shortest paths algorithm by Pettie and Ramachandran [17], this step takes $O(mn^2 \log \alpha(m, n))$ time. (We have n graphs in total since the computed solution uses $n - 1$ edges.)

In the remaining of this section, we fix an edge e, and $\mathbf{r}_{-e}$, and an algorithm $A_i = \text{CENTER}_f$, and we show how to compute the thresholds $\theta_e^i(\mathbf{r}_{-e})$, $\hat{\theta}_e^i(\mathbf{r}_{-e})$ and $\check{\theta}_e^i(\mathbf{r}_{-e})$ in $O(n\,\alpha(n, n))$ time. This implies Theorem 3 since there are m algorithms and n agents/edges e used by the computed solution.

At the heart of the proof is an efficient method for computing, for any edge $f = (u, v)$, the following function in $O(n\,\alpha(n, n))$ time[4]:

$$\hat{F}(\ell) := \inf\{x |\ h_f^*(x, \mathbf{r}_{-e}) \geq \ell\},$$

which is important for us since, as we will see below, $\theta_e^i(\mathbf{r}_{-e}) = \hat{F}(\overline{g}_i)$, $\hat{\theta}_e^i(\mathbf{r}_{-e}) = \hat{F}(g_{\min})$, while the other threshold can be computed similarly.

Let $\delta_{u,v}(G, \mathbf{r})$ be the distance from node u to node v in a graph G with weights $\mathbf{r}$. We consider the following function (let $\inf\{\emptyset\} = \infty$):

$$\hat{f}_z(\lambda) := \inf\{x |\ \delta_{c,z}(G, (x, \mathbf{r}_{-e})) \geq \ell, \text{ where } c = (f, \lambda)\}.$$

Is is easy to see that $h_f^*(x, \mathbf{r}_{-e}) < \ell$ if and only if there exists a $\lambda \in [0, 1]$ such that $\hat{f}_z(\lambda) < \ell$ for *all* $z \in V$. Conversely, $h_f^*(x, \mathbf{r}_{-e}) \geq \ell$ if and only if for *every* $\lambda \in [0, 1]$, there exists $z \in V$ such that $\hat{f}_z(\lambda) \geq \ell$. Let $\hat{f}(\cdot)$ be the lower envelope of functions $\{\hat{f}_z(\cdot)\}_{z \in V}$. It then holds that $\hat{F}(\ell) = \sup_{\lambda \in [0,1]} \hat{f}(\lambda)$.

We now focus on the "shape" of the shortest path distance from c to node z. When node e has weight x, this distance is

$$\delta_{c,z}(G, (x, \mathbf{r}_{-e})) := \min\{\lambda r_f + \delta_{u,z}(G, (x, \mathbf{r}_{-e})), (1 - \lambda)r_f + \delta_{v,z}(G, (x, \mathbf{r}_{-e}))\} \tag{12}$$

which is the minimum of the following two functions:

$$upath_\lambda(x) := \lambda r_f + \min\{x + \delta_{u,z}(G, (0, \mathbf{r}_{-e})), \delta_{u,z}(G - e, \mathbf{r}_{-e})\}; \tag{13}$$

$$vpath_\lambda(x) := (1 - \lambda)r_f + \min\{x + \delta_{v,z}(G, (0, \mathbf{r}_{-e})), \delta_{v,z}(G - e, \mathbf{r}_{-e})\}. \tag{14}$$

These two functions are of the form $\lambda r_f + \min\{x + a, b\}$ and $(1 - \lambda)r_f + \min\{x + a', b'\}$, respectively (see Fig. 1(left)).

The key observation is that, in a configuration like the one in Fig. 1(left), the intersection point with the horizontal line corresponding to ℓ gives a *lower bound* on the values x such that $\delta_{c,z}(x, \mathbf{r}_{-e}) \geq \ell$: the minimum of these two functions is indeed the shortest path distance from $c = (f, \lambda)$ to z and, on the right of the intersecting point, both functions are above ℓ.

More formally, we relate $\hat{f}_z(\lambda)$ to $upath_\lambda(\cdot)$ and $vpath_\lambda(\cdot)$ via their "inverses":

$$upath_\ell^{-1}(\lambda) := \inf\{x |\ upath_\lambda(x) \geq \ell\}; \quad vpath_\ell^{-1}(\lambda) := \inf\{x |\ vpath_\lambda(x) \geq \ell\}.$$

It is easy to see that $\hat{f}_z(\lambda) = \max\{upath_\ell^{-1}(\lambda), vpath_\ell^{-1}(\lambda)\}$. We then observe that these two functions can be described by the "movement" of the intersection point when increasing the value of λ at unitary speed: function $upath_\lambda(x)$ moves upward with speed r_f and thus, as long as it intersects ℓ, $upath_\ell^{-1}(\lambda)$ decreases by r_f (see Fig. 1(left)). For λ sufficiently large, function $upath_\lambda(x)$ is fully not below ℓ and $upath_\ell^{-1}(\lambda) = 0$, while for λ sufficiently small $upath_\ell^{-1}(\lambda) = \infty$ since $upath_\lambda(x)$

[4] We tacitly assume that f is other than e, since the case in which e and f coincide can be easily handled, given that the position of the relative center $c = (f, \lambda)$ returned by CENTER$_f$ does not depend on $x = r_f$.

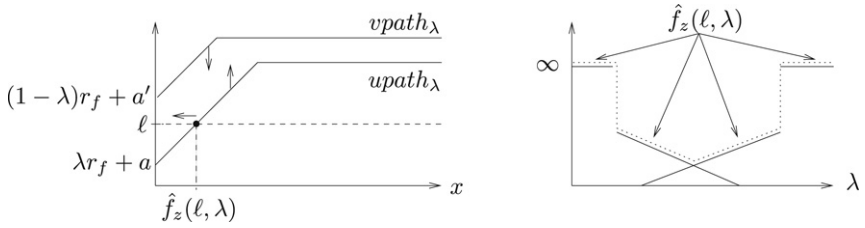


Fig. 1. From shortest path distances to upper envelopes.

is fully below ℓ. The behavior of $vpath_\ell^{-1}(\lambda)$ is symmetric, thus implying that $\hat{f}_z(\cdot)$ can be described by two segments as shown in Fig. 1(right). (Basically, the upper envelope of the two functions.) These segments can be easily computed from the values a b, a' and b' in (13) and (14). The latter can be trivially obtained from the pre-computed shortest path distances in $G-e$. (E.g., $\delta_{u,v}(G,(0,\mathbf{r}_{-e})) = \delta_{u,u'}(G-e,\mathbf{r}_{-e}) + \delta_{v',v}(G-e,\mathbf{r}_{-e})$, for $e=(u',v')$.) Since $\hat{F}(\ell) = \sup_{\lambda\in[0,1]}\hat{f}(\lambda)$, we simply need to compute the lower envelope of the n functions $\hat{f}_z$ which intersect pairwise in at most one point (each function $\hat{f}_z$ is represented by two segments as in Fig. 1(right)). This computation requires $O(n\,\alpha(n,n))$ time once we have computed the segments of each function [1].[5] We have thus shown how to compute $\hat{F}(\ell)$ for each ℓ in $O(n\,\alpha(n,n))$ time, after a pre-processing for computing all-pairs shortest paths in $(G,\mathbf{r})$ and in $(G-e,\mathbf{r}_{-e})$.

From the first step of CENTER$_f$ and from (2) we obtain the following two identities, respectively: (i) $\theta_e^i(\mathbf{r}_{-e}) = \inf\{x|\ h_f^*(x,\mathbf{r}_{-e}) \geq \overline{g}_i\} = \hat{F}(\overline{g}_i)$; (ii) $\hat{\theta}_e^i(\mathbf{r}_{-e}) = \inf\{x|\ h_f^*(x,\mathbf{r}_{-e}) \geq \overline{g}_{\min}\} = \hat{F}(\overline{g}_{\min})$. If we replace '$\geq\ \ell$' with '$>\ \ell$' in the above definitions, we obtain a function $\check{F}(\ell)$ such that $\check{\theta}_e^i(\mathbf{r}_{-e}) = \check{F}(g_{\min})$. (This will only affect whether the slanted segments in Fig. 1(right) are left/right open/closed.)

To complete the proof we show how to compute all $\overline{g}_i$ and $g_{\min}$ in $O(mn\,\alpha(n,n))$ time. Observe that $\overline{g}_i = h_f^*(\infty,\mathbf{r}_{-e})$ for $A_i = \text{CENTER}_f$. In Lemma 1 we showed that CENTER$_f$ runs in $O(n\,\alpha(n,n))$ time if the (precomputed) distances in G are given. We need $O(mn\,\alpha(n,n))$ time to compute all $\overline{g}_i$ and $g_{\min} = \min_i\{\overline{g}_i\}$. After that, each of the $O(mn)$ thresholds can be computed in $O(n\,\alpha(n,n))$ time, and the overall running time is thus $O(mn^2\,\alpha(n,n))$, which dominates the $O(mn^2\log\alpha(m,n))$ factor for the initial distance computations. □

Now observe that the time τ_μ to evaluate the cost (i.e., the height) of a tree is $O(n)$, the number of algorithms in the 'MIN' is $k=m$, and the number N of agents/edges used in any solution is $n-1$. Hence, Theorems 1–3 imply the following:

Corollary 1. *There exists an $O(mn^2\,\alpha(n,n))$-time truthful mechanism for the MDST problem.*

4. Conclusions

We have described a general approach for building truthful mechanisms running in *strongly polynomial time* based on the 'MIN' operator defined by Mu'Alem and Nisan [12]. This is similar to what Kao et al. [10] propose, though their method for computing the payments assumes that each function $g_{A_i}(x)$ is *monotonically increasing* in $x < \theta_e(\mathbf{r}_{-e})$ (see the assumptions preceding Theorem 10 in [10]). For certain optimization problems, this is too restrictive, as it is the case for the MRST and MDST, whose objective functions do not fulfill this requirement, and thus payments obtained from [10] do *not* guarantee truthfulness (in Example 1, their approach would ignore the tie-breaking rule among the algorithms).

Our technique has a very natural application to the MDST problem where the underlying algorithm in [8] optimizing the diameter $d(\cdot)$ can be rewritten as a 'MIN' combination of m algorithms optimizing a *different* function $h(\cdot)$, i.e., the height of a SPT rooted at the relative center of an edge. Quite similarly, it can be shown that for the MRST problem our method yields a mechanism which improves the $O(mn\sqrt{n}+n^3\log n)$ running time in [18] to $O(mn\sqrt{n}+n^3)$. Besides these two problems, we plan to apply our method to develop a strongly polynomial-time truthful mechanism for the *p-center* graph problem [19] (with p fixed, since otherwise the problem is known to be NP-hard), where in addition to the location of the p centers, one has to compute the associated trees.

Although the results have been presented for mechanism design graph problems, they apply to a more general framework in which the agent valuations are either 0 or t_e, that is, to the *known single minded bidders* in [12] or, equivalently, to the *binary demand games* in [10]. The fact that we require plateau-like algorithms (instead of bitonic ones in [12]) does not directly prevent from optimal solutions (any bitonic algorithm minimizing the function $\mu(\cdot)$ is automatically plateau-like). Voluntary participation is guaranteed if optimal algorithms must drop an agent when its cost becomes too high.

An interesting future direction is to apply our technique to NP-hard problems to obtain truthful approximation mechanisms (this was done in [10] for problems maximizing the *welfare*, i.e., the sum of all agents costs which obviously meet the "monotone increasing" requirement). According to Theorem 1, it suffices to show that an approximation algorithm is MIN-reducible in polynomial time. An interesting question here is whether the approximation ratio of the "best" approximation polynomial-time algorithm can be attained by some truthful polynomial-time mechanism.

[5] The technique in [1] assumes the functions to be continuous in the interval of interest. Nevertheless, this technique can be easily adapted to piecewise continuous functions, as those in Fig. 1(right).

Finally, notice that our positive results cannot be extended to the case in which an agent owns several edges of a graph (these problems can model certain scheduling problems for which no exact truthful mechanism exists [15,2], while an extension of Theorem 1 would imply such an exact mechanism).

References

[1] P.K. Agarwal, M. Sharir, Davenport–Schinzel Sequences and their Geometric Applications, Cambridge University Press, New York, 1995.
[2] A. Archer, E. Tardos, Truthful mechanisms for one-parameter agents, in: Proc. of the Annual IEEE Symposium on Foundations of Computing, FOCS, 2001, pp. 482–491.
[3] E.H. Clarke, Multipart pricing of public goods, Public Choice (1971) 17–33.
[4] J. Feigenbaum, S. Shenker, Distributed algorithmic mechanism design: Recent results and future directions, in: Proceedings of the 6th Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, DIALM, ACM Press, 2002, pp. 1–13.
[5] T. Groves, Incentive in teams, Econometrica 41 (1973) 617–631.
[6] L. Gualà, G. Proietti, A truthful (2-2/k)-approximation mechanism for the Steiner tree problem with k terminals, in: Proc. of the 11th Int. Computing and Combinatorics Conference, COCOON, in: LNCS, vol. 3595, 2005, pp. 90–400.
[7] L. Gualà, G. Proietti, Efficient truthful mechanisms for the single-source shortest paths tree problem, in: Proc. of the 11th International Euro-Par Conference, EURO-PAR, in: LNCS, vol. 3648, 2005, pp. 941–951.
[8] R. Hassin, A. Tamir, On the minimum diameter spanning tree problem, Information Processing Letters 53 (2) (1995) 109–111.
[9] J. Hershberger, S. Suri, Vickrey prices and shortest paths: What is an edge worth?, in: Proc. of the 42nd Annual IEEE Symposium on Foundations of Computing, FOCS, 2001, pp. 252–259.
[10] M.-Y. Kao, X.-Y. Li, W. Wang, Towards truthful mechanisms for binary demand games: A general framework, in: Proc. of ACM Electronic Commerce, EC, 2005.
[11] E. Koutsoupias, C.H. Papadimitriou, Worst-case equilibria, in: Proc. of Annual Symposium on Theoretical Aspects of Computer Science, STACS, in: LNCS, vol. 1563, 1999, pp. 404–413.
[12] A. Mu'Alem, N. Nisan, Truthful approximation mechanisms for restricted combinatorial auctions, in: Proc. of 18th National Conference on Artificial Intelligence, AAAI, 2002, pp. 379–384.
[13] R. Myerson, Optimal auction design, Mathematics of Operations Research 6 (1981) 58–73.
[14] E. Nardelli, G. Proietti, P. Widmayer, Finding the most vital node of a shortest path, Theoretical Computer Science 296 (1) (2003) 167–177.
[15] N. Nisan, A. Ronen, Algorithmic mechanism design, in: Proc. of the 31st Annual ACM Symposium on Theory of Computing, STOC, 1999, pp. 129–140.
[16] C.H. Papadimitriou, Algorithms, games, and the internet, in: Proc. of Annual ACM Symposium on Theory of Computing, STOC, 2001, pp. 749–753.
[17] S. Pettie, V. Ramachandran, Computing shortest paths with comparisons and additions, in: Proc. of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms, SODA, 2002, pp. 267–276.
[18] G. Proietti, P. Widmayer, A truthful mechanism for the non-utilitarian minimum radius spanning tree problem, in: ACM Symposium on Parallel Algorithms and Architectures, SPAA, ACM Press, 2005, pp. 195–202.
[19] B.C. Tansel, R.L. Francis, T.J. Lowe, Location on networks: A survey. Part I: The p-center and p-median problems, Management Sciences 29 (1983) 482–497.
[20] W. Vickrey, Counterspeculation, auctions and competitive sealed tenders, Journal of Finance (1961) 8–37.